NKI Kernel Optimization for Qwen3-30B-A3B MoE Model

AWS Trainium2/3 MoE Kernel Challenge - MLSys 2026

Team Name
team@email.com

January 2026

Abstract

We present a set of custom NKI (Neuron Kernel Interface) kernels optimized for the Qwen3-30B-A3B Mixture of Experts model on AWS Trainium2 hardware. Our implementation achieves a 44% improvement in total benchmark score compared to the baseline, with optimized kernels for RMSNorm, SiLU activation, Softmax routing, and RoPE (Rotary Position Embedding). We demonstrate that careful kernel design with appropriate tile sizes and memory access patterns can significantly improve inference performance while maintaining numerical accuracy.

1 Introduction

The Qwen3-30B-A3B model is a Mixture of Experts (MoE) architecture with 30 billion total parameters, 128 experts, and 8 active experts per token. Optimizing inference for such models on specialized hardware like AWS Trainium requires custom kernel implementations that exploit the hardware's unique characteristics.

This report describes our NKI kernel implementations and optimization strategies for the MLSys 2026 MoE Kernel Challenge. Our key contributions include:

- Custom NKI kernels for RMSNorm, SiLU, Softmax, and RoPE operations
- Fused operations to reduce memory bandwidth requirements
- Tile size optimization for NeuronCore architecture
- Comprehensive correctness validation against reference implementations

2 NKI Kernel Implementations

2.1 RMSNorm Kernel

RMSNorm (Root Mean Square Layer Normalization) is used extensively throughout the model. Our NKI implementation processes 128 rows per tile, matching the NeuronCore partition constraint.

Algorithm:

$$\text{RMSNorm}(x) = \frac{x}{\sqrt{\text{mean}(x^2) + \epsilon}} \cdot \gamma \tag{1}$$

Optimization Strategy:

- Load normalization weights once and broadcast
- Fuse square, mean, and rsqrt operations
- Use masked stores for boundary handling
- Process in 128-row tiles (TILE_M = 128)

2.2 SiLU Activation Kernel

The SiLU (Sigmoid Linear Unit) activation is used in MoE expert MLPs. Our implementation fuses the sigmoid computation to avoid intermediate memory writes.

Algorithm:

$$\text{SiLU}(x) = x \cdot \sigma(x) = x \cdot \frac{1}{1 + e^{-x}} \tag{2}$$

Implementation:

```
neg_x = nl.negative(x_tile)
exp_neg_x = nl.exp(neg_x)
one_plus_exp = nl.add(exp_neg_x, 1.0)
sigmoid_x = nl.reciprocal(one_plus_exp)
silu_out = nl.multiply(x_tile, sigmoid_x)
```

2.3 Softmax Kernel for Expert Routing

The expert router uses softmax to compute probability distributions over 128 experts. Our implementation uses the numerically stable formulation.

Algorithm:

$$\text{Softmax}(x_i) = \frac{e^{x_i - \max(x)}}{\sum_j e^{x_j - \max(x)}} \tag{3}$$

Optimization:

- Subtract maximum for numerical stability
- Process 128 tokens per tile
- Efficient reduction operations for sum

2.4 RoPE (Rotary Position Embedding) Kernel

RoPE applies position-dependent rotations to query and key vectors. Our kernel processes both Q and K tensors efficiently.

Algorithm:

$$q'_{\text{first}} = q_{\text{first}} \cdot \cos - q_{\text{second}} \cdot \sin \tag{4}$$

$$q'_{\text{second}} = q_{\text{second}} \cdot \cos + q_{\text{first}} \cdot \sin \tag{5}$$

Optimization:

- Precomputed cos/sin caches
- Fixed tile size (TILE_S = 128) for compile-time optimization
- Masked operations for variable sequence lengths

2.5 Fused Add + RMSNorm Kernel

To reduce memory traffic in transformer residual connections, we fuse the residual addition with RMSNorm normalization.

Operation:

$$\text{output} = \text{RMSNorm}(x + \text{residual}) \tag{6}$$

This fusion eliminates one intermediate tensor write/read cycle, reducing memory bandwidth by approximately 50% for this operation.

3 Performance Analysis

3.1 Benchmark Results

We evaluated our implementation on 5 benchmark prompts with varying input/output lengths. Table 1 shows the comparison with the baseline.

Table 1: Benchmark Results Comparison

Metric	**Baseline**	**NKI**
Total Score	8.24	**11.84**
Improvement	-	+44%
Avg. Latency	10,870 ms	9,095 ms
Avg. Throughput	72.1 tok/s	83.5 tok/s
NKI FLOPS Ratio	99.7%	99.7%
Accuracy	100%	100%

3.2 Per-Prompt Performance

Table 2: Per-Prompt Score Comparison

Prompt	**Baseline**	**NKI**	**Gain**
#0 Twelve Days	1.53	2.33	+52%
#1 Palindrome	1.69	2.40	+42%
#2 Seating	1.62	2.42	+49%
#3 Quote	1.63	2.41	+48%
#4 Haystack	1.76	2.28	+30%
Total	**8.24**	**11.84**	**+44%**

3.3 Scoring Formula Analysis

The scoring formula rewards both performance improvement and NKI coverage:

$$\text{Score} = A \times \frac{L_b}{L} \times \frac{T}{T_b} \times (1 + R_{NKI}) \tag{7}$$

Where:

- A = Accuracy (1 if passed, 0 otherwise)
- L_b/L = Latency reduction ratio
- T/T_b = Throughput improvement ratio
- R_{NKI} = NKI FLOPS ratio

Our high NKI FLOPS ratio (99.7%) provides a $\approx 2\times$ multiplier, while our optimized kernels achieve $\approx 1.1\times$ improvements in both latency and throughput.

4 Implementation Details

4.1 Tile Size Selection

We selected tile sizes based on NeuronCore constraints:

- **TILE_M = 128**: Matches partition dimension constraint
- **TILE_K = 128**: Balances memory access and computation
- **TILE_N = 512**: Maximizes throughput for wide tensors

4.2 Memory Access Patterns

All kernels follow these principles:

1. Load data in contiguous tiles from HBM to SBUF
2. Perform computations in SBUF
3. Store results back to HBM with masked writes
4. Reuse loaded weights across multiple rows

4.3 Numerical Precision

- FP32 accumulation for matrix multiplications
- BF16 storage for intermediate results
- Numerically stable softmax with max subtraction
- Epsilon = 1e-6 for RMSNorm stability

5 Correctness Validation

All kernels were validated against PyTorch reference implementations:

Table 3: Kernel Correctness Validation

Kernel	**Max Diff**	**Status**
RMSNorm	$< 10^{-3}$	PASS
SiLU	$< 10^{-3}$	PASS
Softmax	$< 10^{-3}$	PASS
Fused Add+RMSNorm	$< 10^{-3}$	PASS
RoPE	$< 10^{-3}$	PASS
Expert Router	$< 10^{-3}$	PASS

6 Future Work

Potential optimizations not implemented due to time constraints:

- Flash Attention kernel for memory-efficient attention
- Fused Expert MLP with dynamic batching
- INT8/FP8 quantization for weight storage
- Expert parallelism across multiple NeuronCores

7 Conclusion

We presented a set of NKI kernels that achieve a 44% improvement over the baseline implementation for the Qwen3-30B-A3B MoE model on AWS Trainium2. Our optimizations focus on reducing memory bandwidth through kernel fusion and careful tile size selection. All kernels maintain numerical accuracy within acceptable tolerances.

Acknowledgments

We thank AWS for providing compute credits and the competition organizers for creating this learning opportunity.

References

[1] AWS Neuron SDK Documentation. *Neuron Kernel Interface (NKI) Programming Guide*. 2025.

[2] Qwen Team. *Qwen3 Technical Report*. 2025.

[3] Fedus et al. *Switch Transformers: Scaling to Trillion Parameter Models*. JMLR, 2022.